Introduction

Introduction

The process of speech recognition comes naturally to humans since the ear acts as an auditory system, giving us the gift of sound. The ear converts the mechanical vibration signal of sound into electric signals that the brain can identify. However, computers cannot translate signals naturally like the human ear so we have to come up with creative methods to determine what was said. Speech recognition is complex, but we can simplify it if we only focus on recognizing vowels. Consonants generally have erratic frequency responses, but vowels are special in that their frequency responses have several well defined peaks which are called formants. Long story short, if we can figure out the formants of a vowel, we can figure out the vowel.Insert paragraph text here.

Motivation

Many big name companies like Microsoft, Apple, and Google are creating speech recognition programs. We decided that we wanted to make something useful and relevant to the outside world and build an interactive software. This way our project has more functionality and the user is not restricted when interfacing with the program. Building smooth communication between humans and computers has been a focus of engineering in the 21st century and we wanted to do our part in this incredible, ongoing effort.

Objective

Our team aims to take a small aspect of speech recognition, developing a vowel recognition algorithm and test its accuracy. Our initial objective is to use the Fourier Transform to extract an input vowel signal and have the output identify the vowel. Once that proved mostly successful, we moved on to a harder task: recognizing vowels at arbitrary spacing with arbitrary length. Namely, we challenged ourselves to find a way to interpret the vowels in an stream of speech.

Vowel Recognition Overview

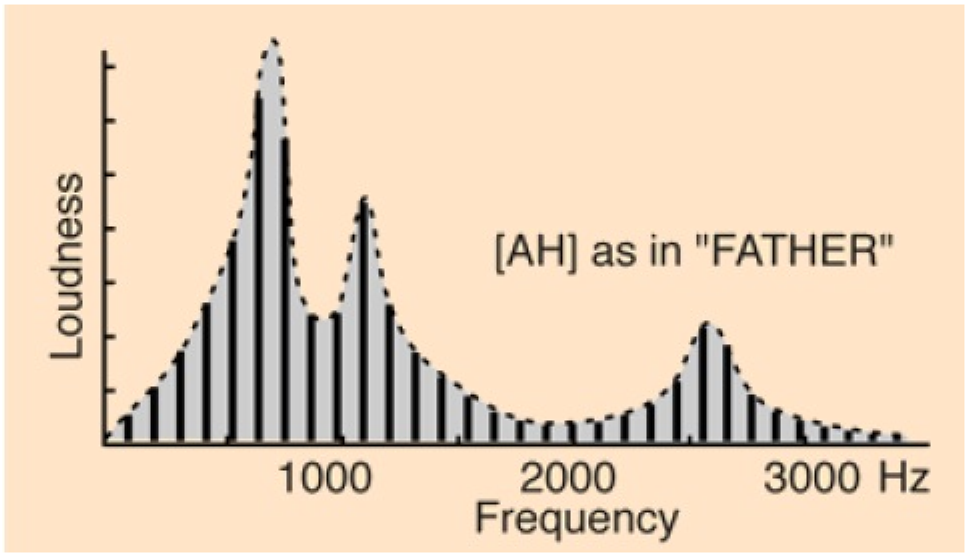
Concept of Vowel Recognition

Vowel recognition is an interesting challenge because it applies system theory to our own physiology. Whenever we attempt to speak, the glottis - the part of the larynx containing the vocal cords - starts to vibrate. These vibrations can be modeled as white noise which become audible and coherent sounds as they pass through the vocal tract. Past experiments have repeatedly confirmed that the vocal tract can be treated as a linear, time-invariant system.

With this, we can proceed in one of three ways: we can model the system as having all zeros (moving average), all poles (autoregressive), or some combination of poles and zeros. Since we can only observe the output of the filter - the speech the escapes the vocal cavity - we choose to model the system as having only poles, because such a model has little dependence on the original input signal. With an autoregressive model, we can generate a transfer function to approximate the filter with a degree of precision proportional to the filter's parameter. It should be noted that a higher order model generally works better but is also more computationally expensive. Once we have the transfer function, we look at the frequency response and determine which frequencies the peaks occur. These frequencies are the formants and we can look at known formant charts to determine which vowel was spoken.

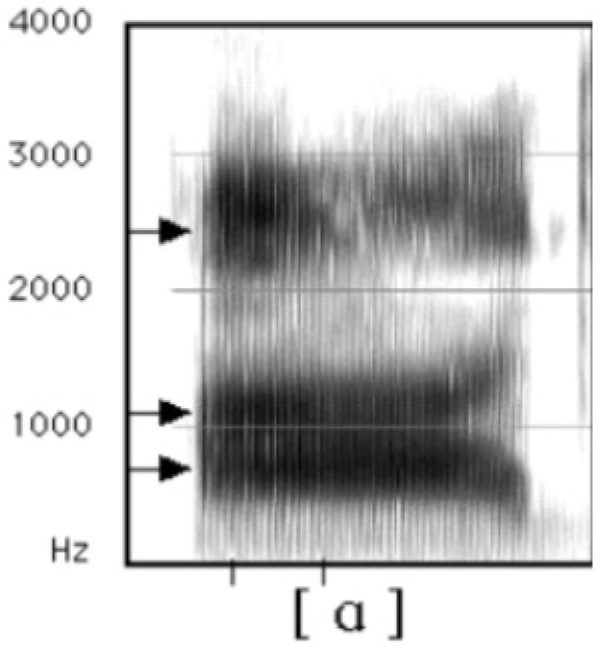
Identifying Vowel Formants

The term formant refers to peaks in the harmonic spectrum of a complex sound. You can see this spectrum by taking a Fourier transform and looking at the frequency response of the signal. Formants in the sound of the human voice are particularly important because they are essential components in the intelligibility of speech. The distinctness of the vowel sounds can be attributed to the differences in their first three formant frequencies. Producing different vowel sounds amounts to returning these formants within a general range of frequencies depending on the particular person.



Formants of an 'Ah' Sound (Adopted from HyperPhysics)

From the frequency response of the vowel formants, we can look at how the peaks of the frequency in the harmonic spectrum line up with the corresponding dark lines in the spectrogram. The dark areas are where the formants are and the graph show them at the same frequency.



Spectrogram of the 'Ah' Sound

The Autoregressive Model

The Autoregressive Model

Speech recognition also utilizes probability theory to understand the effects of the vocal tract system. Since there are many physiological factors that contribute to the buzzing of the glottis, the Central Limit Theorem allows us to model it, , as an iid process with Gaussian distribution of mean zero and constant variance.
Equation:

Knowing that the signal value at any time t is some random variable, we naturally begin to wonder about the autocorrelation of the process, the measure of how correlated signal values at different times are. For independent processes, we know that the autocovariance (which is the same as the autocorrelation in this case) equals zero for any time with some other , and equals the variance at .
This is illustrated in the derivation below for the zero-mean case:
Equation:

Equation:

Equation:

With this in mind, we know that
Equation:

We now know the autocorrelation of the input and the autocorrelation of the output (the output is known - it's the speech signal). We define the DTFT of to be the power spectral density of x. That is,
Equation:

We can show that for LTI/LSI systems, the power spectral density of the output becomes the power spectral density of the input multiplied by the square of the magnitude of the transfer function. Note that this holds only for wide-sense stationary processes - i.e., processes for which is only dependent on and not on . Starting with the convolution sum,

Equation:

Equation:

Equation:

Equation:

Equation:

Letting , this becomes:

Equation:

Equation:

Equation:

Equation:

So now we can solve for the transfer function. Note that the value of the variance does not matter, because it affects the magnitude of the graph but does not affect its shape. The system is depicted below.

Equation:

If we use a linear, time invariant system with only poles, we can model the output as a recursive average as shown.

Equation:

Statisticians often refer to a model of the following form as an autoregressive process. Note that it has the same form as our equation above.

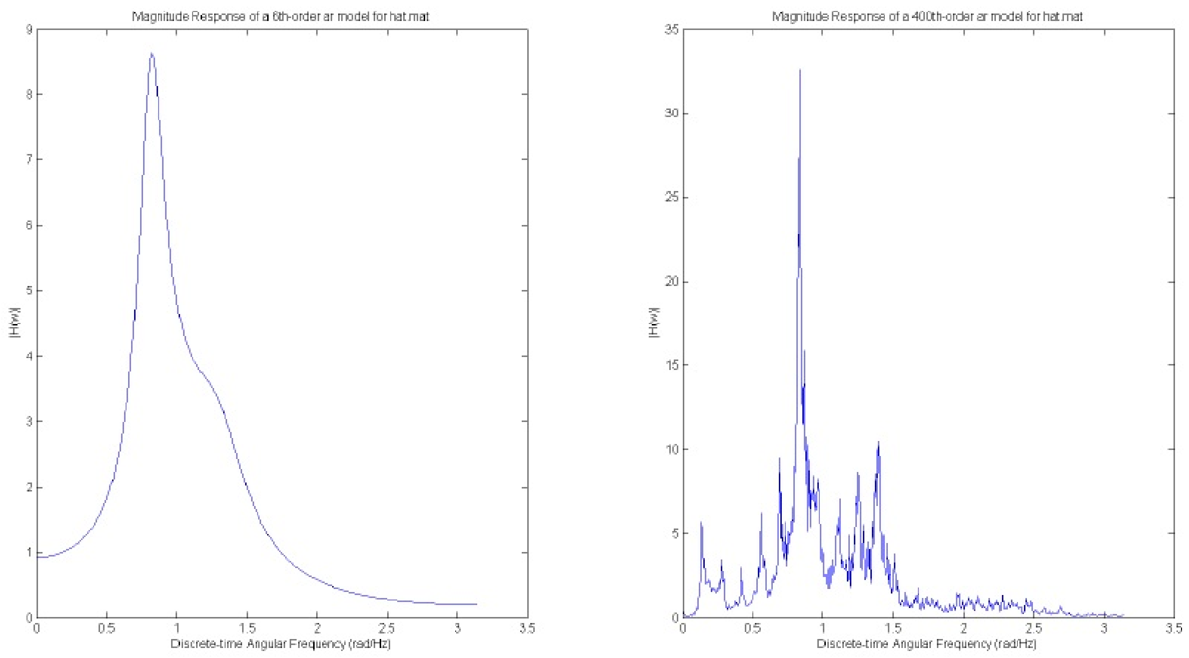
Equation:

They have algorithms which can find the values of that constant. Those constants are the coefficients of the polynomial in the denominator of the transfer function. This is essentially the autoregressive model we talked about earlier. The input is the numerator of our transfer function. Again, the actual value of the numerator doesn'‾™t matter because it only changes the magnitude of the frequency response. Now that we know how to model the transfer function, we can tackle the formants.

One Vowel Recognition

After we gathered information about probabilistic interpretations of speech recognition, we were ready to begin building the project. Starting with the basics, we built a simple program that would take as input a three-second long signal and produce as output a measure of the corresponding formants of the signal. By applying the auto-regressive model, we were able to come up with a reasonable transfer function modeling the speech. However, we encountered a slight problem.

If we estimated the transfer function using an auto-regressive model of low order, the associated frequency response would be a little too smooth. I.e., it would perhaps mask some of the peaks of the graph and cost us valuable information in determining the formants. On the other hand, if we used a high-order model, we would get too much variation in the frequency response because of the intricacies of the high-order rational function. We would not be able to tell which peaks were true formants and which peaks were just "overestimation" by our model. This is demonstrated in the figure below for the test speech signal "hat.mat".

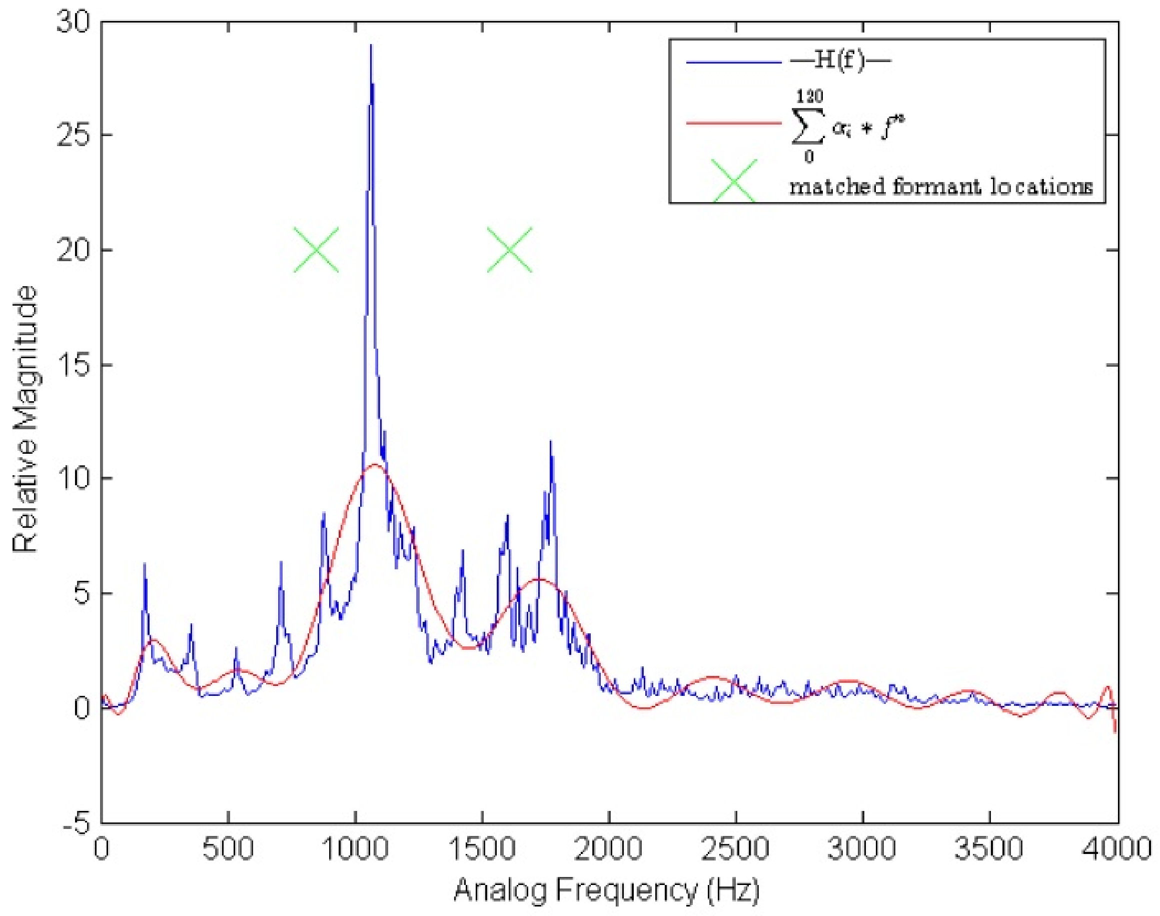


Low vs High Order of the Autoregressive Model

This eventually started to look like an optimization problem, and since none of us had much relevant expertise, we approached it slightly differently. Instead of solving a case-by-case parameter optimization problem with unnecessarily difficult math, we simply abstracted the problem in terms of something we knew - polynomial regression.

The Taylor series tells us that we can approximate every function as a polynomial, whose order determines its degree of precision. As with the parameter for the AR model, increasing the polynomial approximation's order would result in a high degree of information saved from the signal. Since we were more familiar with Taylor series than machine learning and statistical approximation, we chose to rely more strongly on polynomial regression than the AR model.

With this in mind, we used MATLAB to solve a generalized least-squares regression problem. In other words, we came up with a solution that is both accurate and precise. We used a high-order (30 order) a.r. model to get the accuracy we needed and a moderate-order(120 order) polynomial regression to get the precision we needed. This is illustrated in figure below for "hat.mat".

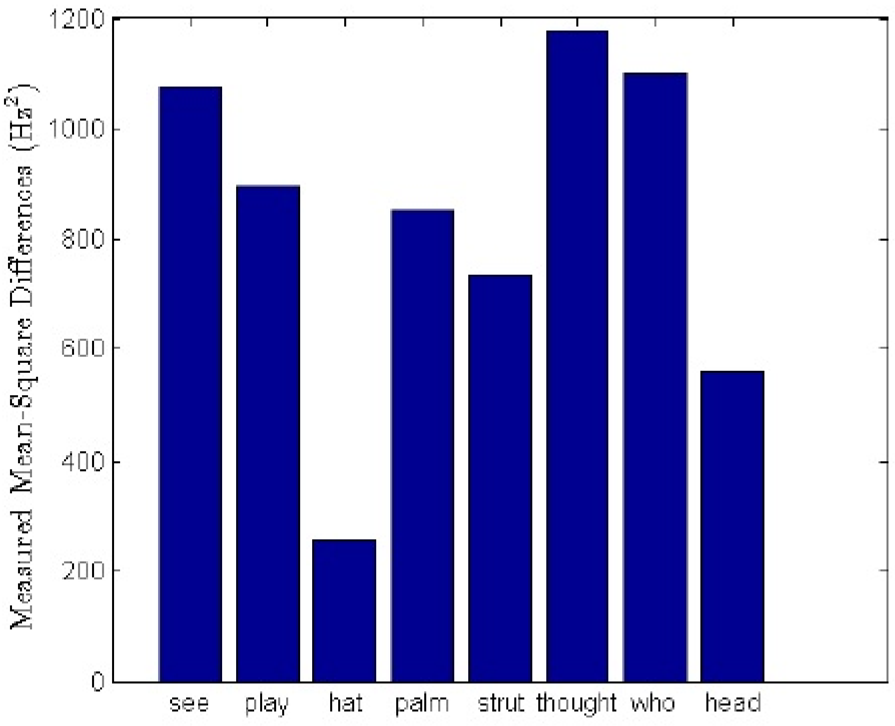


Formant Determination of the Signal when Sujay is Saying the Word 'Hat'

After using predetermined average vowel formants found on the internet to estimate the formants of the signal, we then implemented a matched-filter which would find the vowel whose theoretical formant values were most similar. Our filter was based on the method of least-squares, so it chose the vowel for which the mean squared difference was smallest. We plotted the measures of the mean-squared differences for hat.mat below. Note that the lower the bar, the more similar the match between the corresponding vowel and the signal.

$$\frac{1}{2}\sum_{1}^{2}(f_{measured} - f_{theoretical})^2$$

Mean Squared Difference Equation



Measured Mean-Squared Differences of the Signal when Sujay is Saying the Word 'Hat'

Multiple Vowel Recognition Analysis

The Thought Process

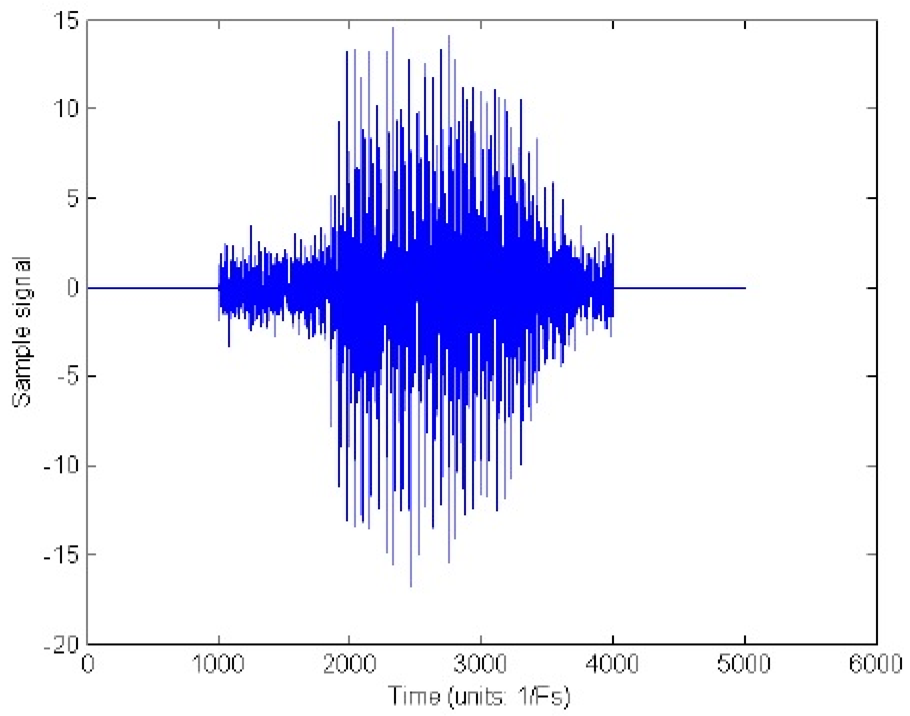
Initial Considerations

The program we initially developed was capable of accepting one vowel sound over a course of one or two seconds. Naturally, we wished to expand our program capacity to work for sequences of vowels with variable separation in time. With the increasing complexity of our project, however, we faced even more situations and problems we had to consider. The first and most important problem we had to consider was whether or not to parse the data into sections where we would "guess" the location of the vowel. The next problem, closely connected with the first, was a method of differentiating noise from actual sound content. A last problem was implementing the method optimally into MATLAB.

For reasons that will become clear soon, we chose to compute vowels by creating a continuous, nonoverlapping partition of the signal instead of just separating out vowel content.

The Math

Consider the sample signal below. The actual vowel content is contained within samples 2000-3500. White Gaussian noise has been added between samples ~1000-4500. This superposition represents a combination of both external noise infiltrating the signal and unwanted content in the speech itself. To clarify that last part, consider the following example. In determining the vowel in "hat," we must find a way to deal with 1) external noise, 2) the consonants 'h' and 't', and 3) the transition between a consonant and the vowel and vice-versa.



Sample Signal of the Word 'Hat' with Problematic Noise

If we go the path of taking the chunk between samples 1000 and 4000 and guessing it's the vowel (which is the best we can do in the "parsing" case), we take into consideration a lot of unwanted noise that will affect our formant guesses. Since roughly half the power in this signal is noise, our results may be corrupted. How might we get around this?

We looked to a partitioning method instead, because it seemed noise-effective to a higher degree. Consider taking the entire signal and dividing it into chunks of 500 samples a piece.

Let's say that we iterate through the sequence of chunks and put the following restriction on our method: "Whenever we see that four chunks of our signal match to the same vowel's formants, we say that vowel is definitively in the signal." If we proceed this way, we notice that samples from 1000 to 2000 will yield two formant pairs that *might* be similar, while the samples between 2000 and 4000 yield four very similar formant pairs because of the vowel content. Since the formants identified in the initial noise don't satisfy the initial restriction, we throw them out. Since the formants identified in the signal itself do satisfy the restriction, we keep them as a means to identifying the vowel. While time-"parsing" can't really deal with the complication of noise, it is easy to see that with this new "partition" method, we are effectively filtering out the effects of inconsistent noise (both external and internal) by throwing out any unreliable data.

This helps get rid of some of the potential inaccuracy of our program, but how might we deal with inevitable background noise? We determined experimentally that the mean (absolute value of the) amplitude of any partition containing the speech content is orders of magnitude higher than parts that are just background noise. With that, we set two conditions that would automatically exclude any such noise from being in our consideration. As we analyze the signal in progressive chunks, we first look to see if the chunk has a max amplitude of at least 0.1 of the max amplitude of the signal. Then, we check to see that the mean of the magnitude of the chunk was greater than or equal to the mean of the signal. If one or more of these conditions were not met, we would immediately discard the current chunk and move on to the next chunk of signal.

The Application

We built the prototype by dividing our problem into five subproblems: initializing our data, reading an audio signal, determining the frequency response, cleaning up formant data and displaying information relevant to the vowels.

Initial Data

In our function initialize_all_data, we store the theoretical vowel formant pairs for the words "see", "play", "hat", "palm" and "rug" into a matrix for later use. We also set up a convenient matrix called the output_matrix, created so that we could elude the need to go through 7 if-else statements when outputting a vowel.

Audio Input
After the user decides how long he or she wants to speak, we use a built-in MATLAB script to read in an audio signal of that length from the user. The sampling rate of 44.1kHz is intentionally high so that we can preserve a lot of information from the original signal.

Assumptions
1) The user will speak at a relatively constant amplitude. Note that this is a little easier said than done, because words like "see" are naturally much quieter than "hat."

2) The user will speak slowly and clearly so that we can look for consistency in determining the vowels said.

The Algorithm
1) Establish a set of variables that store theoretical vowel formants.

2) Record an audio signal of n seconds from the user.

3) Split the data into non-overlapping chunks of 4000 samples.

4) Preprocess the data by detrending with a low-order polynomial and using a low-order Butterworth lowpass filter.

5) Determine the transfer function of the vocal tract associated with the current chunk using an ar model.

6) Determine the peaks of the transfer function (the formants) and match with the closest formant, filtering by a least means-squared matched filter.

7) If amplitude of signal exceeds 0.1 max amplitude of overall signal, we assume it's potentially a vowel. Put its formants onto the "stack" of recent_formant_pairs.

8) If the last four formant pairs have been consistent (the same value), then we will assume the vowel estimation has worked. Add the formant pair to the "stack" of vowels identified in tracktimes_and_formants.

9) Now we have processed our guesses for what vowels were said when. We are going to have repeats, so run through a for-loop to clean up the track_times_and_formants vector into a new, workable vector called track_begin_end_formants.

10) Output data and guesses.

Limitations

1) Requires very clear enunciation. Difficult to establish because we naturally change the pitch of our words as we start and end them. Example: the end of the vowel in "strut" sounds remarkably similar to "hat" because of the way you shape your vocal tract as you close your mouth.

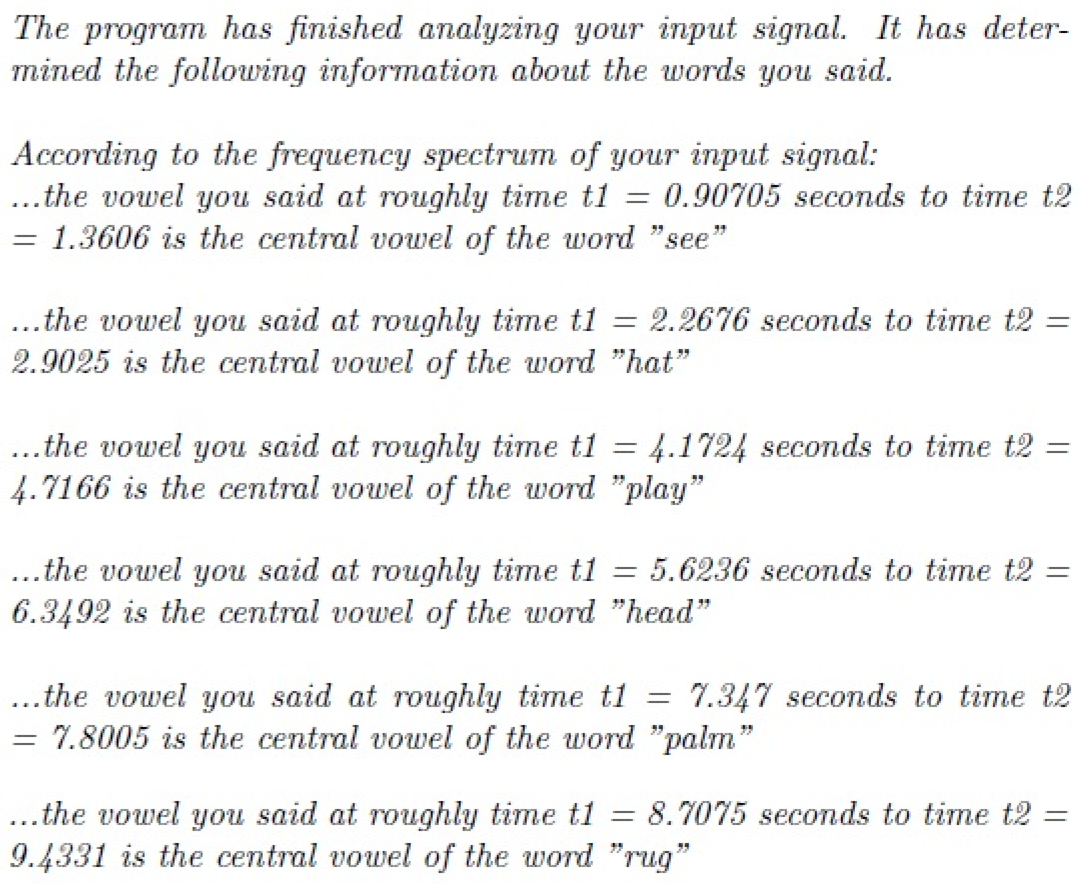
2) User has to say each vowel for a moderate duration of time. Too long, and a wavering voice would affect success of the program. Too short, and not enough data to assign formants.

3) This program is calibrated with formant values averaged across all ages and genders. If a male has an exceptionally deep voice, or a child/female has an exceptionally high voice, they may not be able to get accurate vowel readings from the program. Accents may also affect accuracy of results.

Data Collection and Analysis

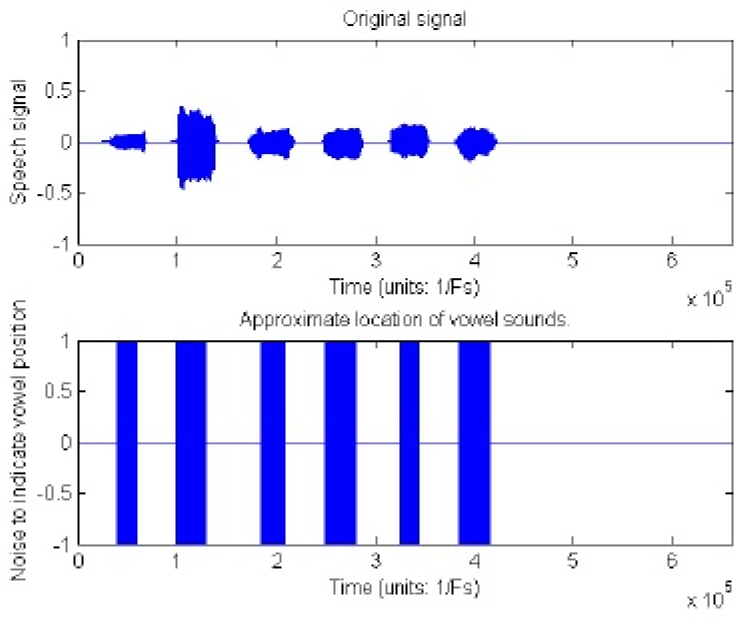
Procedure
When we began testing of our program, we knew that we would have to be very clear in conveying its strengths and limitations from the start. Otherwise, users might be tempted to rush and speak incoherently in the excitement to see if the program worked. With this in mind, we gave the following set of instructions to every person who tested our program. First, they needed to speak clearly and slowly, extending their vowels just a little beyond their usual length. Next, they were allowed to add pauses of any gap between the words as long as the pause was noticeable. Most importantly, any consonants they pronounced had to be soft, not hard. In the word "rug," for example, the hard 'g' gets a lot of emphasis and ends up overshadowing the rest of the word (including the vowel). That's why we told them to shift their normal pronunciation of "rug" to a softer "ru-ck." After these precautions were given, the user would say some combination of one-syllable words similar to the ones in our predefined database, waiting to see that the program might recognize this aspect of his voice.

Results
Our program was very successful, ranging from nearly perfect with shorter one-to-three word speeches, to moderately accurate with four-to-six word speeches. To see just how effective the program is, let's run through its capabilities with a test signal. Click here for a sample of Sujay speaking the words "see", "hat", "play", "head", "palm" and "rug". Since we anticipated handling both live speech and stored audio files, we run our program with the name of the file as a parameter and wait. Shortly, MATLAB produces the following output.

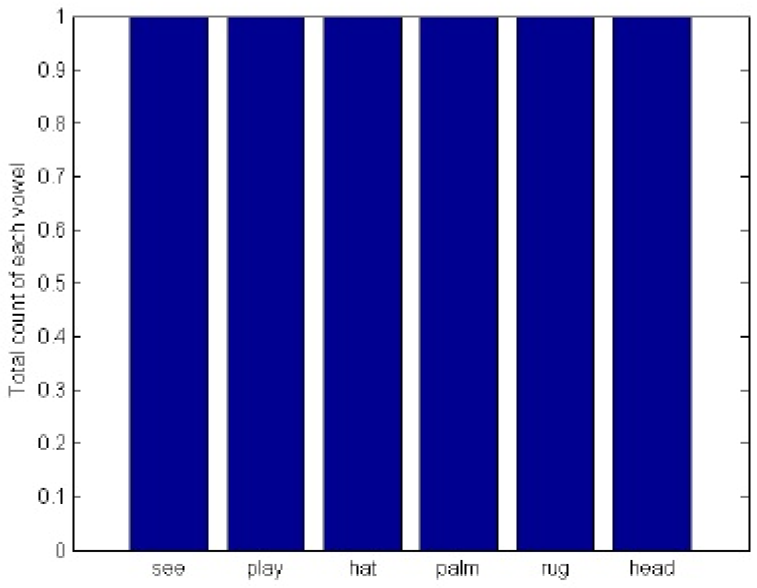


```
The program has finished analyzing your input signal. It has deter-
mined the following information about the words you said.

According to the frequency spectrum of your input signal:
...the vowel you said at roughly time t1 = 0.90705 seconds to time t2
= 1.3606 is the central vowel of the word "see"

...the vowel you said at roughly time t1 = 2.2676 seconds to time t2 =
2.9025 is the central vowel of the word "hat"

...the vowel you said at roughly time t1 = 4.1724 seconds to time t2 =
4.7166 is the central vowel of the word "play"

...the vowel you said at roughly time t1 = 5.6236 seconds to time t2 =
6.3492 is the central vowel of the word "head"

...the vowel you said at roughly time t1 = 7.347 seconds to time t2
= 7.8005 is the central vowel of the word "palm"

...the vowel you said at roughly time t1 = 8.7075 seconds to time t2 =
9.4331 is the central vowel of the word "rug"
```

At the very least, this tells us that the program can recognize the vowels chronologically. But how can we ensure that the program identified the correct time ranges for the vowels? Along with the text shown, the program also outputs a graph showing where in time it estimates sound content in the original signal. It is shown in the figure below. Interestingly enough, by avoiding the "parsing" approach to this problem, we managed to do an even better job of parsing the signal into its vowels.



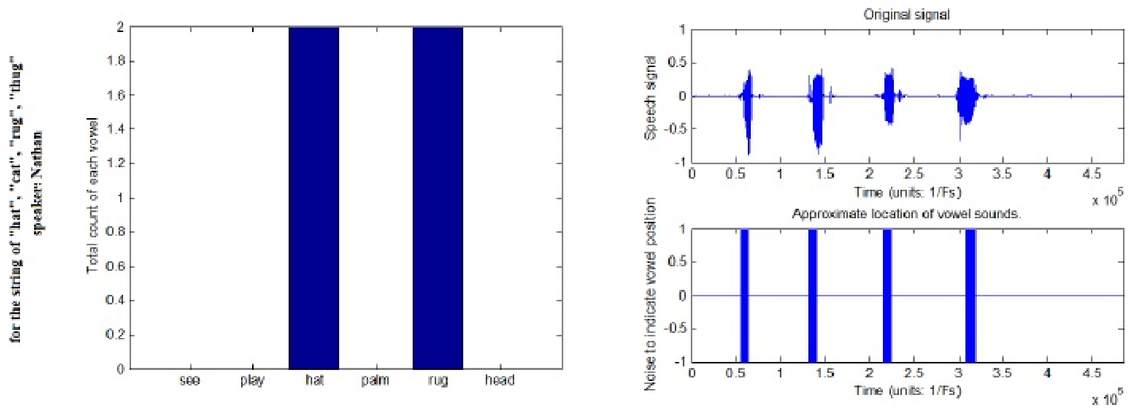
And, for good measure, a total count of each vowel is displayed to visualize the program output for longer strings of words.



More Data

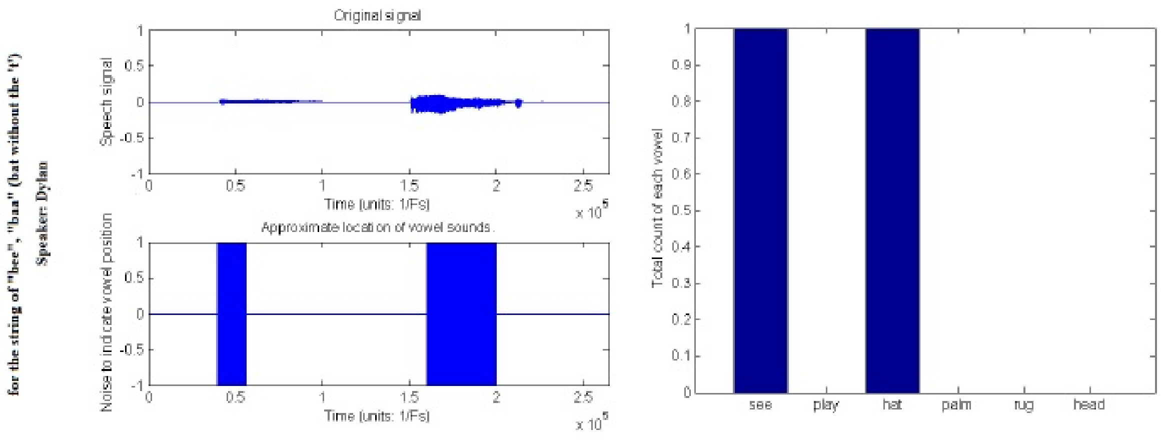
A few more samples of data are shown below for demonstrative purposes.

Figure 9: Nathan Bucki says "Hat Cat Rug Thug"



...the vowel you said at roughly time t1 = 1.2699 seconds to time t2 = 1.4513 is the central vowel of the word "hat"
...the vowel you said at roughly time t1 = 2.9932 seconds to time t2 = 3.1746 is the central vowel of the word "hat"
...the vowel you said at roughly time t1 = 4.898 seconds to time t2 = 5.0794 is the central vowel of the word "rug"
...the vowel you said at roughly time t1 = 6.9841 second to time t2 = 7.2563 is the central vowel of the word "rug"

Figure 10: Dylan Jones says "Bee Baa" (pronounced like bat)"



...the vowel you said at roughly time t1 = 0.9075 seconds to time t2 = 1.2699 is the central vowel of the word "see"
...the vowel you said at roughly time t1 = 3.6281 seconds to time t2 = 4.5352 is the central vowel of the word "hat"

Data Analysis

We found that our project was generally successful, but our data collection showed us some flaws that we did not anticipate. First and foremost, the natural tonal variation in pronunciation made it difficult to determine vowel content. Typically when one says the word "see," they start off with a higher pitch and end with a lower pitch. Since our program works best for monotonous vowel sounds - that is, vowels whose sounds do not vary from one time to the other - it was challenging to succeed without first warning the user to maintain monotone in the course of every vowel. Second, we found that however hard we tried to throw out the effects of consonants on our program, they still appeared, albeit to a small degree. In the formation of the a word beginning with "h", for example, one starts off by slightly parting the lips and blowing air before proceeding to say the rest of the word. If the person is trying to say "hat," he or she would maintain his output of air while stretching his mouth and tightening his vocal cords. If he or she is trying to say "head," he or she opens his or her mouth vertically while applying similar pressure to his vocal cords. This phenomenon of speech is known as coarticulation. Note that although the end result - the vowel - is different from "head" to "hat", the process of getting there is very similar. That's why our program sometimes mixed up words that start the same, like "head" and "hat", and words that end the same, like "hat" and "strut". Last but not least, we found that the program did not work as well for people with accents or exceptionally deep/high voices. This was a little frustrating, but not unexpected, because even Google and other big companies struggle with this issue regarding its audience.

Instructions for Usage

1. Download the file here and unzip the files to a directory of your choice.

2. Modify lines 34 and line 41 in main_program.m to reflect the desired directory for data and the current directory, respectively.

3. Open MATLAB and change MATLAB's working directory to wherever you unzipped the files. If you wish to test the program with live audio, run main_program and follow the instructions. If you wish to test the program with prerecorded audio, run Elec301spectrogram_loadfile(parameter), where parameter is the name of your matlab .m audio file in single quotes and without the extension. The audio file must be contained in the same folder as the program's files.

4. Follow the on-screen instructions. Make sure to speak slowly and clearly, with monotonous vowels and distinct pauses between each word.

5. Let the program work its magic! For loaded audio, the full output will be displayed on the console. For live audio, the full output will be available as a pdf file in a child of the data directory that is unique to your run of the program!

Future Work

In the future, we might hope to be able to implement some more effective ways to get rid of consonant effects in testing this program. Or better yet - we could figure out how to process the consonants and develop a guess of what those might be as well! An interesting idea for the current status of the project would be to implement a Markov chain relating a sentence to the vowel flow in that sentence. That way, if a user spoke a short phrase, we could make a guess as to what sentence he said!

Contributions

Sujay: Developed and implemented the algorithm to recognize vowels in a continuous-stream in MATLAB. Wrote about the thought process, coding and data analysis.

Mauro: Researched and implemented the autoregressive model in MATLAB. Wrote about the probabilistic model.

Steven: Researched and implemented functions in MATLAB that were useful for data gathering and analysis. Wrote about the vocal tract system, limitations, and theory of formants.

Project Poster
https://cnx.org/content/m52128/

Conclusion

Speech recognition is an important goal of engineering worldwide, but to enable the broader study of speech processing as a whole, we have to study it in chunks. For this project, we focused on vowel recognition, studying the theory behind formants and random processes that would enable us to build a highly successful program. Our initial development allowed us to recognize one vowel at a time, but upon further inquiry, we were able to implement an intelligent vowel recognition algorithm for streams of isolated words.

References

http://www.sfu.ca/sonic-studio/handbook/Formant.html.

http://hyperphysics.phy-astr.gsu.edu/hbase/music/vowel.html.

http://en.wikipedia.org/wiki/Formant.